Empirical Comparison of Machine Learning Techniques in High Frequency Order Imbalance Based Trading Strategies

by
Cody Cao , Kurtis Lee , Lily Li, Xueliu Wang
CODYC, KURTISL, YANGMINL, XUELIUW

Carnegie Mellon University
MSCF NYC Campus

Contents

1 Introduction

When we consider order driven markets, the primary types of orders are market orders and limit orders. Market orders are executed immediately at the best[1] bid and ask price. Limit orders are posted in the limit order book (LOB). The LOB is a tool that allows traders to submit buy (sell) orders to be executed at pre-specified bid (ask) prices with bid (ask) volumes. [2] As a result, the LOB contains information about the trader's intention to buy or sell at different prices.

If we find that there is more volume posted on the bid side of the book, we would identify that as a positive order imbalance (buying pressure). In this paper, we show that order imbalances are positively correlated with changes in the asset price (mid-point price). This is not a new phenomenon. Positive order imbalance tends to persist in a momentum fashion as traders split their orders across ticks [3]. Therefore, signals from order balances give the trader a chance to make profit on the momentum. Order imbalances have been studied extensively [1] on daily data but there has been much less research done on testing order imbalances on high frequency data. As humans develop the rules and algorithms for bots to trade at low latency, we expect to see a similar predictive effect at that scale.

Empirically, high frequency trading firms (HFT) use linear regression models to forecast price changes, due to their simplicity and efficiency in computation – two highly desirable qualities in the high frequency space.

We extend the work by Shen [2] who tested the merits of order imbalance models in the high frequency space. The analyses from [2] strictly involved linear regression. We first confirm the feasibility of applying linear regression models for order imbalances in the high frequency space. While Shen tested on the Chinese Index Futures (CSI 300) we will test on S&P 500 Futures. We extend Shen's paper by analyzing the merits of advanced techniques at the cost of interpretability, simplicity, and speed. In particular, we compare the four methods: linear regression, support-vector machines, random forests, and random forests with gradient descent (XGBoost).

2 Methodology

2.1 Validation

To ensure robustness and statistical integrity, we preemptively chose to implement this trading strategy as follows:

1. Train on one day's worth of data
2. Test on the next day
3. Move one day forward and repeat

Whether this method of implementation is used in practice would vary by firm and is highly proprietary information.

We held out the first four days (01/02/2019 - 01/07/2019, excluding Sundays and/or days with no activity) to cross-validate our tuning parameters, which varies by estimator. Because we are dealing with time series data, we use a one-day-ahead approach as our cross-validation scheme, identical to strategy implementation. Specifically, this approach yields three models (three instances of train on today, test on tomorrow), for each set of parameters.

The choice of cross-validation window must strike a balance between bias and variance. Observing too much of the data could capture unwanted seasonality shifts in terms of trading patterns and volumes, likely to lead to

[1] Best refers to lowest bid and highest ask.

[2] *Volume* is the amount of a certain contract.

overfitting. Observing too little would leave us with little room to tune. We feel that the four-day window matched our needs well, given that the retraining and testing all occur within a one-day time frame.

2.2 Performance Metrics

2.2.1 Accuracy

Because the trading strategy is reliant on a strong classifier, we must carefully choose our loss function to reflect the needs of the strategy. Our classifier will determine whether an incoming trade opportunity is a buy (+1), hold (0), sell (-1). We consequently define our loss matrix from the perspective of a trader or a general practitioner, shown in Table 1.

Table 1: Loss Matrix

		Predicted		
		Buy	Hold	Sell
	Buy	0	1	2
True	Hold	1	0	1
	Sell	2	1	0

Our loss matrix is defined as such for multiple reasons. Trivially, any correct prediction regardless of which class will be treated as having no loss, whereas incorrect predictions will be punished with some positive loss.

We then note the differences between the following true vs. predicted pairings: buy/sell vs. hold; buy/sell vs. sell/buy. The former, in practice, should net 0 profit or loss (modulo transaction costs) given that future prices are martingales. However, the latter is worse; we are directly contradicting a price movement by predicting the opposite. From this, we would expect substantial detriment to a trader's profit and loss should she make such decisions. To reflect these outcomes, we choose to normalize errors on predicted holds or true holds to be 1; while these may on average net 0 profits or loss, from a statistical standpoint, these are still misclassified. Additionally, any mispredictions for buy with sell or vice versa receives a larger loss of 2.

We also consider two altered versions of accuracy in classification (henceforth referred to as trading accuracy). This consists of correct buy/sell predictions made over the intersection of sets of true buy/sell and predicted buy/sell (Trading Accuracy 2), as well as an augmented version (to include "noise") where we also look at predicted holds over true buy/sell scenarios (Trading Accuracy 1). These scenarios could be treated as missed opportunities from the standpoint of a trader.

2.2.2 Speed

Speed is a concern primarily from a practical point of view. High frequency trading often occurs on the time scale of milliseconds, so any delay in prediction or trading could cost a trader potential arbitrage opportunities. Furthermore, we are not so concerned with how long each model takes to train, as training or retraining occurs outside of trading hours.

To compare prediction speed, we time each estimator's speed (after training) on predicting the signals for two hundred thousand data points held constant for each trial. For robustness, we conduct multiple trials, randomizing the set of test data points. Speed ultimately depends on the complexity of the model both in terms of covariates trained on and underlying structure.

2.3 Models

2.3.1 Linear Regression[3]

Empirically, linear regression has been successful in predicting price changes from order imbalances. A linear relationship is appropriate here as increases in order imbalances typically suggest linear increases in price. Furthermore, linear regression is highly desirable in high frequency trading due to speed in training, execution, and model simplicity.

Because our feature set is relatively small, we can do appropriate variable selection for our model. We fit a linear regression over all features and found co-variants with positive coefficients. Figure 3 summarizes the linear regression models.

Here, we manually select the three variables whose coefficients are 100% positive (in case of buy, similarly negative coefficient in cases of sell) and significant at the 95% confidence level, which are the 2.5-second lagged OI, 2.5-second moving average OI, and 5-second moving average OI. These covariates support our intuition as well.

We then run this model through the trading strategy. After obtaining the $\widehat{y}$, which is the estimated 10-second moving average spread, we converted our response variables (both predicted and true) into the classes $\{+1, 0, 1\}$ using the following computation:

$$\text{signal}_t = \mathbb{1}_{\Delta M_{t,k} \geq Q} - \mathbb{1}_{\Delta M_{t,k} \leq Q} \tag{1}$$

where $\Delta M_{t,k}$ is $\widehat{y}$ and y. Now we have turned our problem back into a classification problem, for the purpose of analyzing the results using confusion matrix, loss functions and accuracy.

Using the ADF test (table 4), we see that 10-second moving average spread $\Delta\overline{M}_{t,20}$, 2.5-second lagged OI ($OI_5$), 2.5-second moving average OI ($\overline{OI}_5$) and 5-second moving average OI ($\overline{OI}_{10}$) all rejected the null hypothesis of having a unit root. We further support the idea that the response variable and explanatory variables are stationary by using the KPSS test (table 5). From the test results, 90% of the time we failed to reject the null hypothesis, supporting stationarity and allowing us to apply the same trading strategy to generate positively increasing profits due to the stationarity of our time-series process.

2.3.2 Support-Vector Machine (SVM)

We suspect that SVM is a good choice of model for a few reasons. Firstly, of the available linear classifiers, SVM assumes the least about our data. For example, linear discriminant analysis requires each class has its data in clumps, and logistic regression requires the log odds have a roughly linear relationship with its covariates; our use of SVM makes no such assumptions. Secondly, predicting with SVM is computationally quick; given the fitted hyperplane, the classifier needs only to evaluate the hyperplane function at a set of coordinates.

We restrict ourselves to linear SVM as using a polynomial or radial kernel would be inefficient computationally – the training time for nonlinear SVM scales superlinearly with the number of samples. To tune our SVM, we primarily focus on tuning our penalty parameter. We perform a grid search over a logarithmically evenly spaced interval from 10^{-5} to 10^5 to select the penalty parameter. The features utilized in SVM include each lagged order imbalance as well as order imbalance ratio and 10- and 20-period moving averages. As the signal classes (buy, hold, sell) are drastically imbalanced, we train the SVM using balanced class weights.

[3] Results and tables can be found in appendix D

2.3.3 Random Forest

Decision trees are an obvious choice for this classification problem. They perform their own variable selection although we currently do not test large sets of features. In addition, random forests improve upon trees and are hard to overfit. Random forests are especially appropriate for this exercise due to the unbalanced nature of the data. As the signal classes (buy, hold, sell) are drastically imbalanced, we train the forest using balanced class weights. We also calculated out-of-bag errors.

One approach to cross-validating over a bigger parameter space is random sampling. We perform randomized search cross validation to regularize and save expensive computational time.

The model was tuned on the cross-validation data on the following parameters:

1. Maximum depth of tree
2. Minimum samples per leaf
3. Minimum samples per split
4. Number of estimators

2.3.4 XGBoost

In addition to why decision trees are appropriate for this model, boosting adds an additional layer of quality. Boosting is appropriate for this exercise due to the unbalanced nature of the data. It also produces lower bias. Additional tuning parameters in XGBoost allow us to modify the model towards our strategy. Balanced weights were attempted but dramatically worsened model performance. As a result, weight balancing was not incorporated in the final model.

The model was tuned on the cross-validation data on the following parameters:

1. Learning rate (η)
2. Maximum depth of tree
3. Minimum child weight
4. Minimum loss reduction
5. Subsample ratio of the training instance
6. Subsample ratio of columns for each level

3 Results

The following is an analysis of Trading Accuracy 1 across all the models[4]. Note that Day 9 and 29 were holidays (Martin Luther King and President's day, respectively) but were included since they were valid trading days [5]. Therefore, all models had anomalous results on those days. In practice, a firm would likely not use the model on those days.

[4] A full summary of the results can be found in section 4.

[5] The lack of trading activity on these days can be seen in figure 28 in the appendix (section F).

[6] Average time in seconds to predict two hundred thousand signals.

Table 2: Results by Estimator.

	Loss		**Trading Accuracy 1**		**Trading Accuracy 2**		**Speed**
	Training	Test	Training	Test	Training	Test	Prediction Time[6]
Linear Regression	12911	12859	21.23%	23.21%	96.39%	96.34%	0.005
SVM	12471	12948	30.64%	32.12%	96.38%	96.31%	0.018
Random Forest	17739	19834	70.88%	52.41%	90.88%	85.98%	2.371
RF with Boost	9529	9559	7.28%	6.55%	95.72%	94.62%	4.850

3.1 Linear Regression

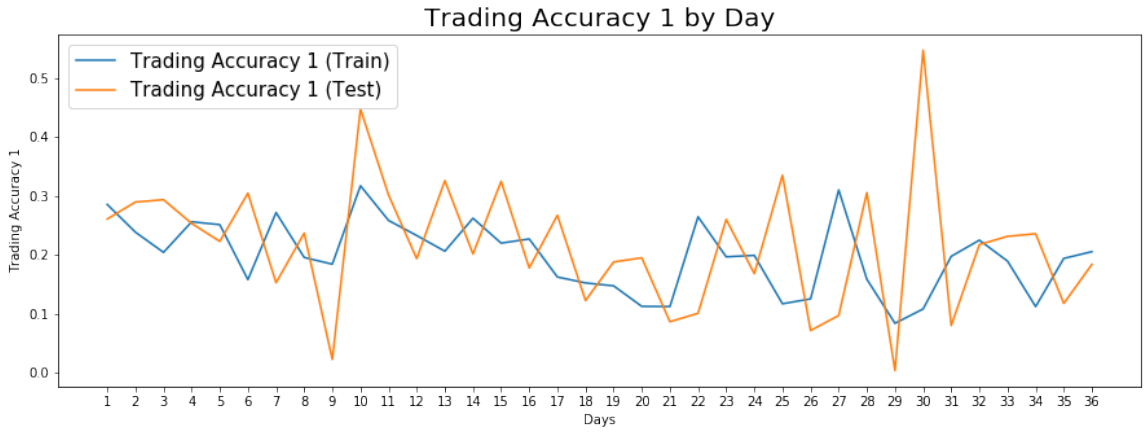

Figure 1: Linear Regression Trading Accuracy 1 by Day

Trading accuracy 1 is mostly constant throughout day 5 to day 42,except on day 23. Further investigation shows that on these days we make little to no trades. This is because the model is rather conservative compared to models like random forests. Specifically, it tends to predict hold signals rather than buy/sell signals to minimize losses. We believe that the high losses yielded from the low liquidity of transactions on these days.

3.2 SVM

SVM trading accuracy 1 is fairly volatile throughout the entire testing period. Noting the differences between figures 2 and 11, there's high volatility in the predictions of the model to hold in place of buying or selling; for the most part, the nominal difference in the two accuracies suggest that our model overwhelmingly predicts for the trader to hold.

Observing that the training and testing accuracy follow one another closely, the models behave similarly across days with similar predictions. We suspect that the model would perform better with a polynomial kernel as these boundaries are potentially highly nonlinear.

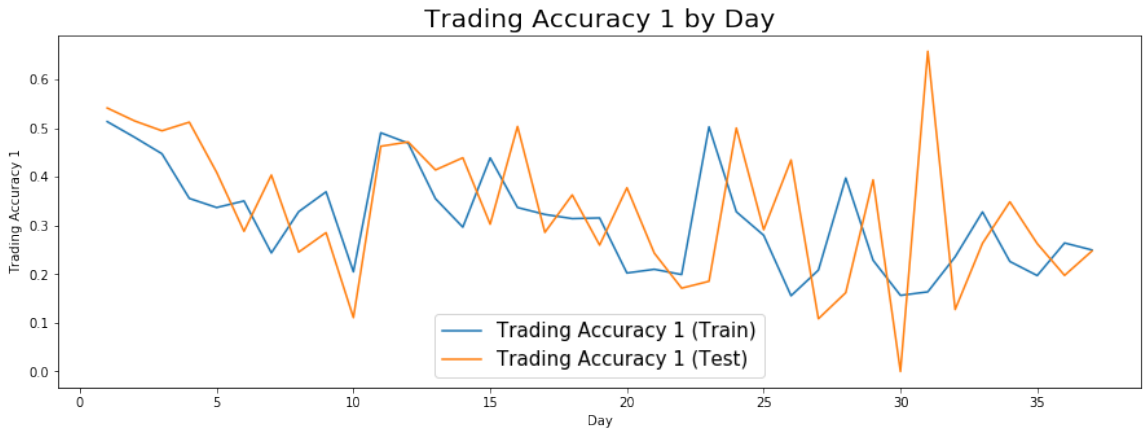

Figure 2: SVM Trading Accuracy 1 by Day

3.3 Random Forest

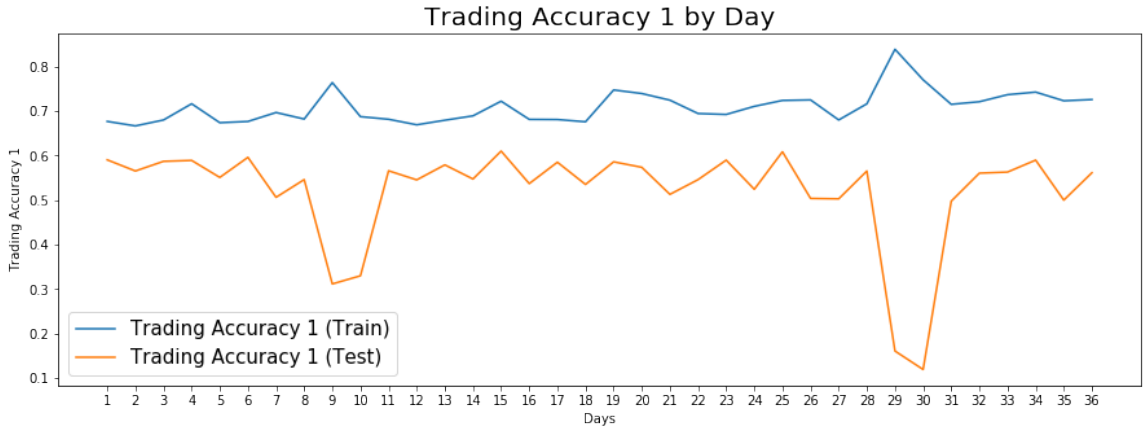

Figure 3: Random Forest Trading Accuracy 1 by Day

The trading accuracy is fairly constant throughout the testing period. We suspect overfitting given the training error dominates the testing error each trading day. We could regularize more by tweaking the range of estimators or investigating more parameters in the random forest. However, the trading and testing error are smaller than the other models. This indicates comparatively aggressive buying/selling as the random forest places relatively smaller weights in the Hold-Buy and Hold-Sell boxes. This is a significant result that differs from the other models.

3.4 XGBoost

The trading accuracy is volatile over time. There are a few days where the trading accuracy is very low, namely day 22, 26, and 35. Further investigation shows that on these days we make little to no trades. The model operates conservatively which was a result of tuning to minimize the loss. It favors making no trade over risking a buy or a

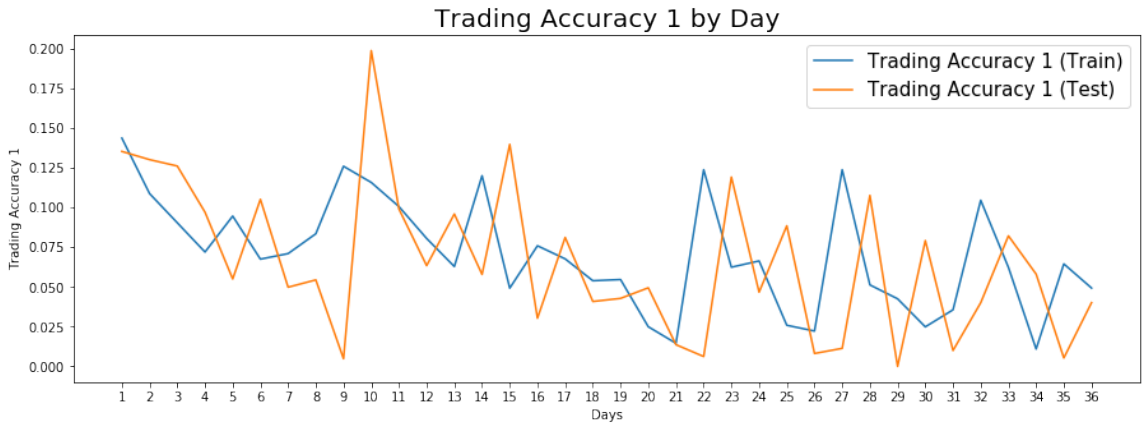

Figure 4: Random Forest XGBoost Trading Accuracy 1 by Day

sell and thus results in anomalous days like this. In particular, it reflects the performance of the model that it was trained on the previous day. This may indicate some form of overfitting or the trading days being very different. This trend generally happens towards the end of the time series as the the future approaches expiration. However, further considerations should do a deep dive into this phenomenon. We also checked for seasonality among days but found no clear trend. In general, since we were not able to successfully account for the imbalanced data in this model, We suspect this is why the trading accuracy was overall low.

4 Conclusion

High frequency order imbalances are still correlated with price changes and can be used to predict trade signals. For Shen's analysis that modeled Chinese futures and ours that modeled S&P 500 futures, we find consistent intra-day correlation on millisecond intervals. Indeed, linear regression is an excellent model for this scenario. Not only does it perform the quickest, but it provides nearly the most optimal performance. SVM performs better on Trading Accuracy 1 (TA1), indicating that SVM models capture the most opportunity. SVM is only marginally slower and it is unlikely for us to lose out on profits due to this speed differential.

The random forest models provide interesting insights. The basic random forest model performs the best on TA1. This is expected since random forests are particularly good at dealing with imbalanced data. But with the highest loss and lowest TA2, it is buying when it should be selling and vice versa. However, had we used a PnL metric it is possible that the random forest model would perform best.

Most notable of the XGBoost model is the minimized loss at the cost of a low TA1. There were several days where the XGBoost model made no trades. The cross validating that minimized loss trained the model to be conservative on trades. As a result, it has a relatively high TA2 but low volume of trades. As expected, it performs the slowest which indicates scalability problems if more features and data were added.

4.1 Future Considerations

There were several limitations in time and complexity we faced while working on this project. We believe that addressing these limitations in further work can expand the validity of the results and provide more interesting

insights into order imbalance modeling:

4.1.1 Availability of Data

Modeling was completed on two months of data. Shen [2] simulated trading over a year. Because we were unable to get high frequency data from any free resources we purchased the data directly from a vendor. We would try to gather more data for a whole year and re-run the modeling on a longer time scale. In general we don't think this was an issue for our results given the large size of data occurring on any given day, but our model performance is potentially susceptible to seasonality in the first quarter of the year.

4.1.2 Trade Simulation

We validated results on a a classification level but not exactly how the model would be used. Ideally the model would be tested in a trading environment, making trades based on the signals. Since we make no assumptions on sizing, we believe our models still provide valuable insight to how well it would perform in a real trading environment. Future projects should consider testing the models in a simulated trading environment.

4.1.3 Comprehensive Trading Strategy

Models were trained solely on classification. Advanced models can be built to determine appropriate sizing of trades and exit rules. Further projects should incorporate these trading strategies into the training and validation process, in tandem with the trading simulation.

4.1.4 Cross Validating the Signal Threshold

We did not cross validate the value of Q (the threshold that separates buy/sell signals from hold signals) but simply used the minimum bid-ask spread. We were limited by time and computational complexity. Future research should explore different values for Q via cross-validation.

4.1.5 Improved Features

Although our simple set of features proved significant and powerful we could certainly test more features given the size of the data. Further projects should consider more complex features to test that might provide further insight into the dynamics of order imbalances. As mentioned during the results analysis, additional features will allow for random forest models to shine and we may find that more advanced models perform better than linear regression.

4.1.6 Data Imbalance

With the exception of the random forest model, our models had a tendency to hold when we needed to buy or sell potentially due to the majority of signals for any given day being hold signals. While the random forest may have circumvented this issue, the other models failed to perform similarly. Future research should attempt to resolve this issue.

References

[1] B. T. MARSHALL E. BLUME, A. CRAIG MACKINLAY, *Order imbalances and stock price movements on october 19 and 20, 1987*, The Journal of Finance, XLIV (1989), pp. 827–848.

[2] D. SHEN, *Order imbalance based strategy in high frequency trading*, Master's thesis, University of Oxford, 2015.

[3] A. S. TARUN CHORDIA, *Order imbalance and individual stock returns*, Journal of Financial Economics, (2002).

A Trading Strategy Assumptions

Our strategy makes several operational assumptions regarding how we would apply the models.

1. Our systems have minimal latency between receiving the data and making the trades. In general, we are able to submit orders within seconds of receiving the data.

2. The market is highly liquid and our orders are fulfilled at the current bid and buy price when an order is submitted.

3. Trading costs are minimal.

Models are constructed at the end of the trading day to be used on the following trading day. At every tick the model recalculates and decides to buy, sell, or hold based on the predicted signal. We make no assumptions regarding sizing, exit rules, or stop loss. Further considerations regarding trading strategy are discussed in the conclusion.

B Data

We acquired two months of E-mini S&P 500 futures contract (ES), with contract start date ranging from 1/1/2019 to 2/28/2019 and expiring in March 2019, from *Tick Data Market*[7]. The original data set provided bid price, bid size, ask price, and ask size on the time order of milliseconds (ms).

We explored the data and found the most active trading hours to be from 8am to 3pm, so we took every weekday's data in these hours, in 500 ms intervals. To organize our data, we regularize our time scale to consider the midpoint price (average of bid and ask) every 500 ms starting from 8AM to 3PM (our trading strategy trades/predicts every 500 ms). Because the order book only reports the orders made and not the continuous price of the contract, we interpolate midpoint prices by taking the most recent price for a given 500 ms time point. We dropped all the rows with missing data, all of which were days with no trading activity.

We also created new variables around order imbalances, midpoint price changes, and signals. Finally, we created the following new variables:[8]

- Midpoint of bid/ask price: M_t
- One period change in midpoint price: dM_t
- Order imbalance (lags and moving averages): OI_t

  Lags: 1 to 5 period

  Moving averages: 5, 10, 20-period
- Order imbalance ratio
- Change in midpoint price metrics: ΔM_t

  Forward moving averages (10 - and 20 - period)

  Moving averages (10 - and 20 - period)

[7] www.tickdatamarket.com

[8] $t = 1, 2, \ldots$ refers to ticks of length 500 ms.

- Signals (buy, sell, hold)

We denote the bid and ask volumes at time t to be V_t^B, V_t^A, respectively; we also denote the best bid and ask prices at time t to be P_t^B, P_t^A, respectively.

B.1 Volume Order Imbalance

We define the following:

$$\delta V_t^B = \begin{cases} 0, & P_t^B < P_{t-1}^B \\ V_t^B - V_{t-1}^B, & P_t^B = P_{t-1}^B \\ V_t^B, & P_t^B > P_{t-1}^B \end{cases}, \qquad \delta V_t^A = \begin{cases} V_t^A, & P_t^A < P_{t-1}^A \\ V_t^A - V_{t-1}^A, & P_t^A = P_{t-1}^A \\ 0, & P_t^A > P_{t-1}^A \end{cases} \tag{2}$$

We then denote *Volume Order Imbalance* (VOI) with OI_t to be:

$$OI_t = \delta V_t^B - \delta V_t^A \tag{3}$$

These definitions are consistent with those found in [2].

B.1.1 Lags

A k-period lag of VOI is defined to be: OI_{t-k}

B.1.2 Moving Averages

$$\overline{OI}_{t,k} = \frac{1}{k} \sum_{i=1}^{k} OI_{t-k} \tag{4}$$

We primarily consider $k = 5, 10, 20$.

B.1.3 Order Imbalance Ratio

$$OI_t^{\text{ratio}} = \frac{V_t^B - V_t^A}{V_t^B + V_t^A} \tag{5}$$

B.2 Midpoint Metrics

B.2.1 Midpoint of bid/ask

$$M_t = \frac{P_t^B + P_t^A}{2} \tag{6}$$

B.2.2 Changes in midpoint price

The one period change in midpoint price is:

$$dM_t = M_t - M_{t-1} \tag{7}$$

B.2.3 Moving Average

$$\overline{M}_{t,k} = \frac{1}{k}\sum_{i=1}^{k} M_{t-k} \tag{8}$$

B.2.4 Forward Moving Average

$$\overline{M}^{F}_{t,k} = \frac{1}{k}\sum_{i=1}^{k} M_{t+k} \tag{9}$$

B.3 Signal

First, we define the following:

$$\Delta M_{t,k} = \overline{M}^{F}_{t,k} - \overline{M}_{t,k} \tag{10}$$

Let $Q > 0$ be our movement threshold (in this case, the tick size for futures prices so $Q = 0.25$). We can define our signal at time t to be:

$$\text{signal}_t = \mathbb{1}_{\Delta M_{t,k} \geq Q} - \mathbb{1}_{\Delta M_{t,k} \leq Q} \tag{11}$$

The signal is what produces our 3 classes for the classification problem (+1, 0, -1). For testing, we use $k = 20$.

C Linear Regression Figures

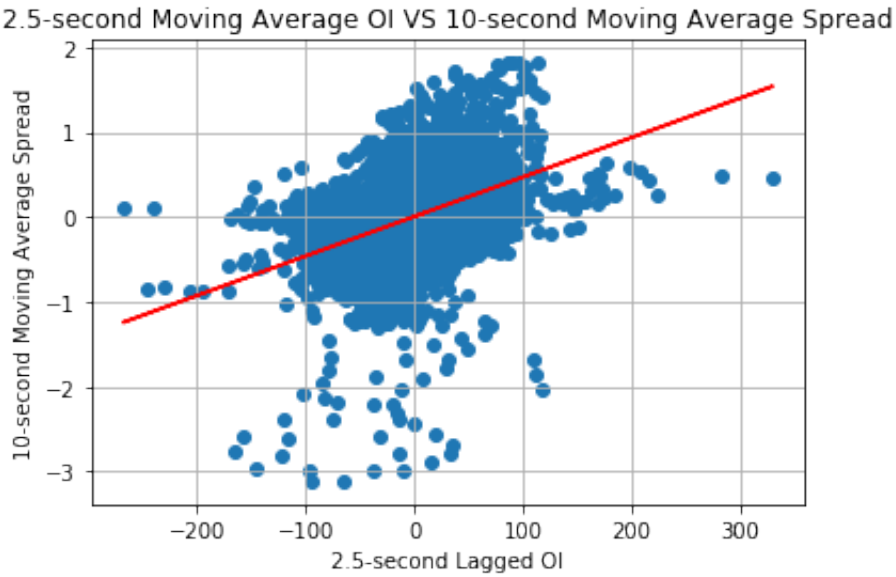

Figure 5: 2.5-second Lagged OI VS 10-second Moving Average Spread

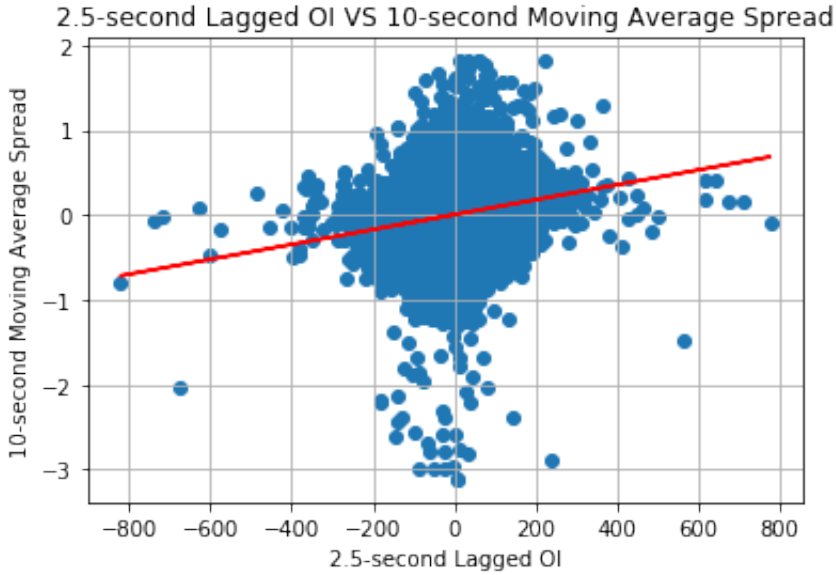

Figure 6: 2.5-second Moving Average OI VS 10-second Moving Average Spread

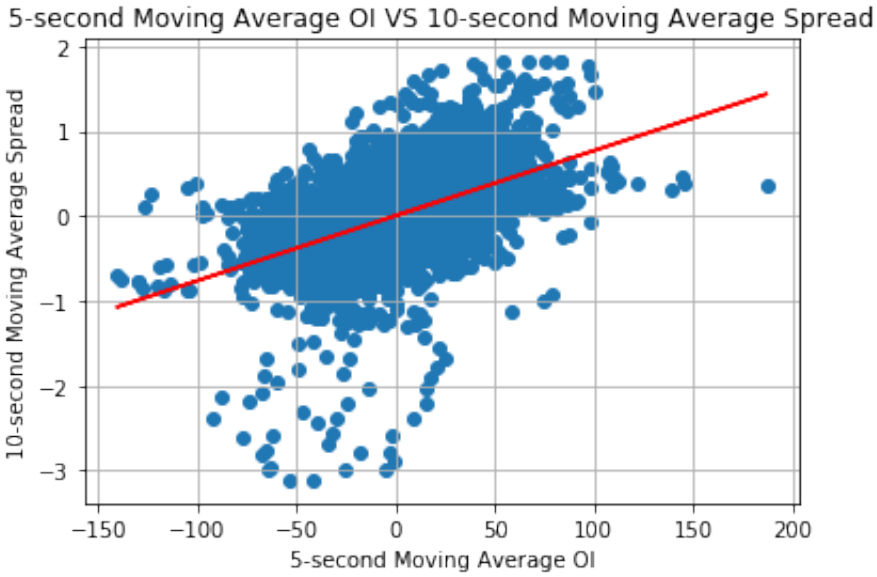

Figure 7: 5-second Moving Average OI VS 10-second Moving Average Spread

D Linear Regression Tables

D.1 Coefficient Table

This table reports coefficients when estimating the change in moving average price. Namely

$$\Delta M_{t,20} = \beta_0 + \sum_{i=1}^{5} \beta_i OI_{t,i} + \beta_6 \overline{OI}_5 + \beta_7 \overline{OI}_{10} + \beta_8 \overline{OI}_{20} \tag{12}$$

Table 3: Regression Results

	Average Coefficient	**Percent Positive**	**Percent Positive and Significant**	**Percent Negative and Significant**
Intercept	6.57×10^{-5}	41%	30%	27%
OI_1	−0.0001270	0%	0%	97%
OI_2	−0.0002035	0%	0%	97%
OI_3	−0.0002773	0%	0%	100%
OI_4	−0.0003601	0%	0%	100%
OI_5	0.00018909	100%	100%	0%
$\overline{OI}_5$	0.00306318	100%	100%	0%
$\overline{OI}_{10}$	0.00483351	100%	100%	0%
$\overline{OI}_{20}$	0.00583308	100%	100%	0%

D.2 ADF Test

H_0: x has a unit root on the 4 days of the data
H_1: x does not have a unit root on the first 4 days of the data

Table 4: ADF Test Results

	$\Delta\overline{M}_{t,20}$	OI_5	$\overline{OI}_5$	$\overline{OI}_{10}$
Reject H_0[9]	100%	100%	100%	100%
Fail to reject H_0	0%	0%	0%	0%

D.3 KPSS Test

H_0: x has a unit root on the 4 days of the data.
H_1: x does not have a unit root on the first 4 days of the data

[9] At the 95% confidence level
[10] At the 95% confidence level

Table 5: KPSS Test Results

	$\Delta\overline{M}_{t,20}$	OI_5	$\overline{OI}_5$	$\overline{OI}_{10}$
Reject H_0[10]	10%	10%	10%	10%
Fail to reject H_0	90%	90%	90%	90%

E Loss Metrics Over Time

We note that test losses overlap training losses due to the imbalance of data in the Hold-hold box of the confusion matrix. Each day's loss is mainly driven by the number of opportunities for profit. Since the Hold-Hold square has most of the volume, we will see similar loss on training and testing each day.

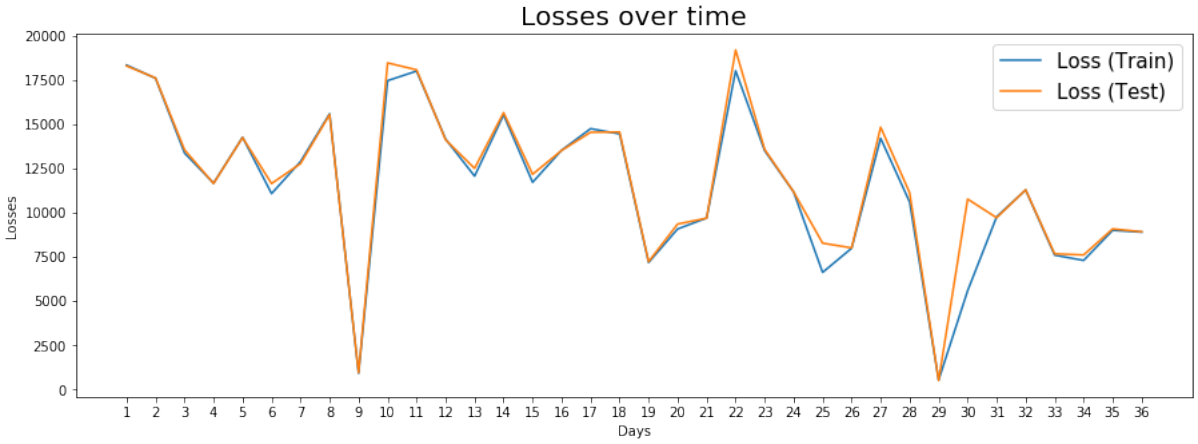

Figure 8: Linear Regression Loss Over Time

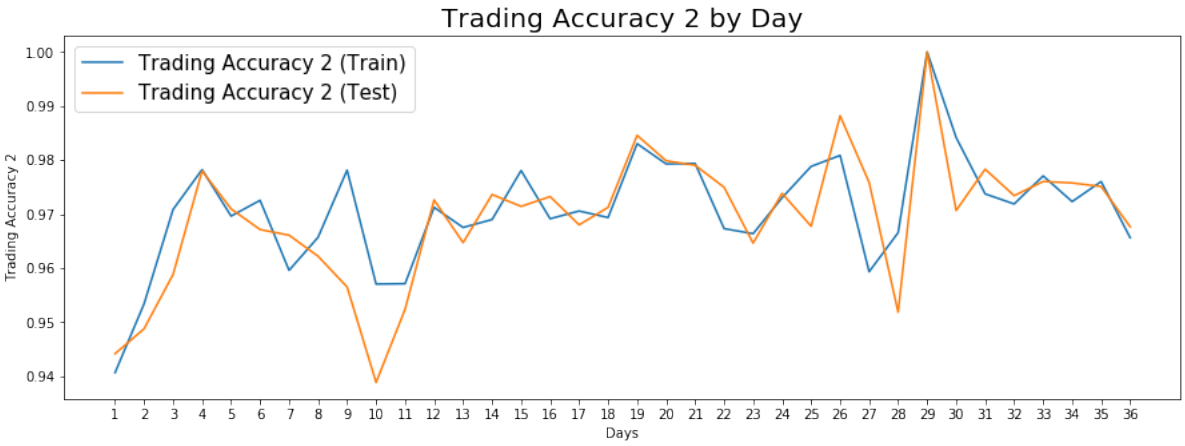

Figure 9: Linear Regression Trading Accuracy 2 Over Time

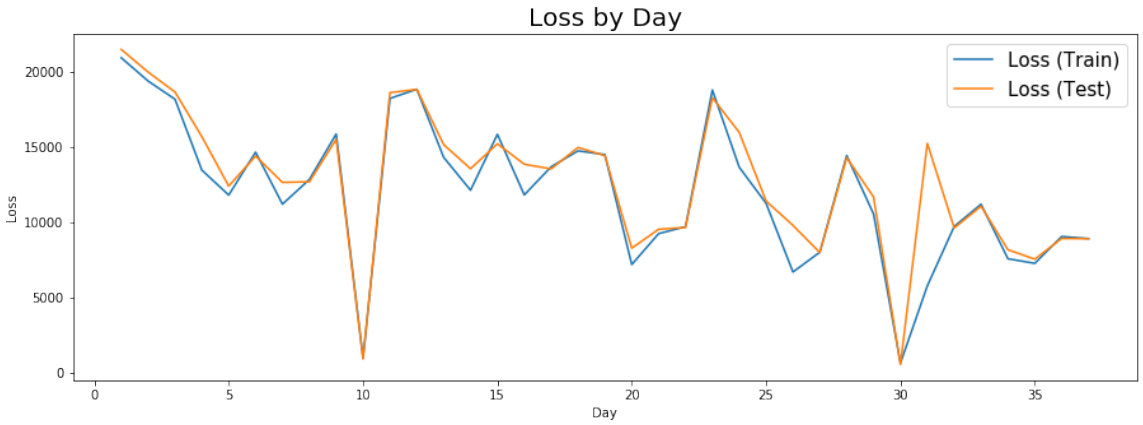

Figure 10: SVM Loss Over Time

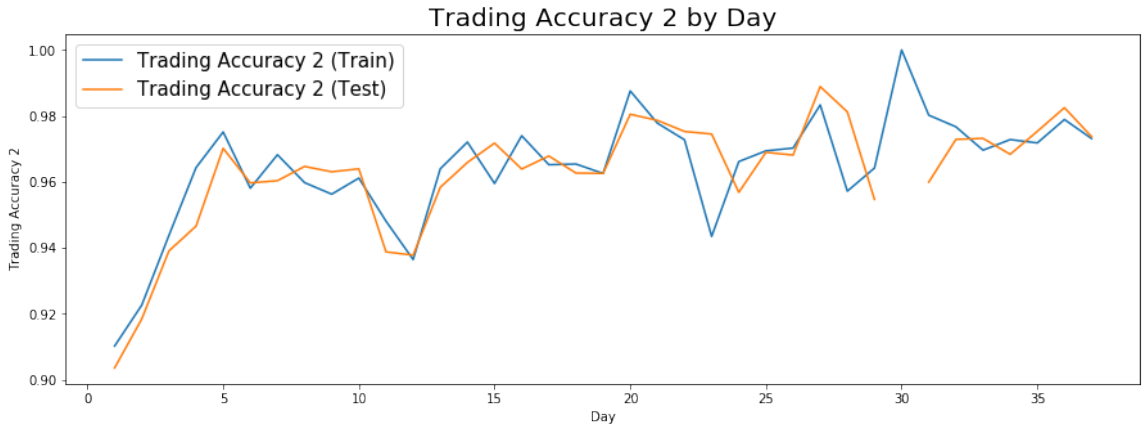

Figure 11: SVM Trading Accuracy 2 Over Time

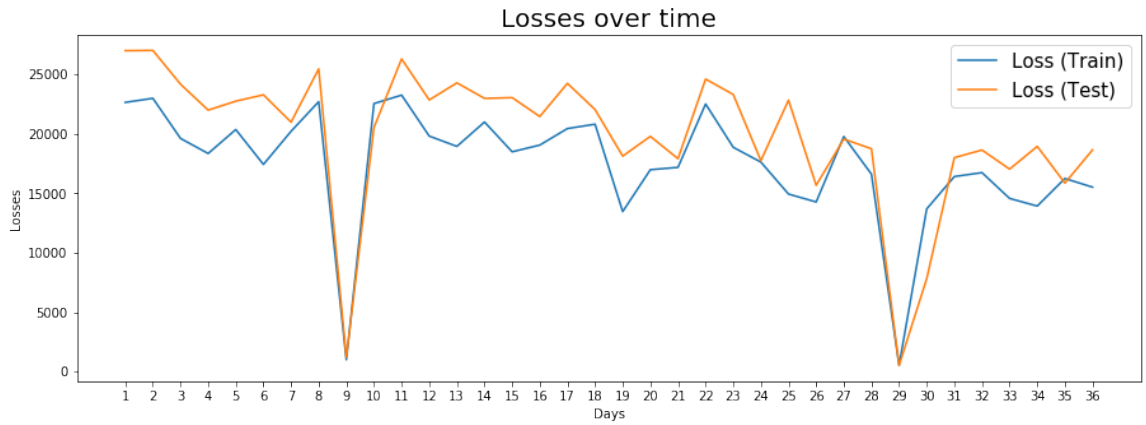

Figure 12: Random Forest Loss Over Time

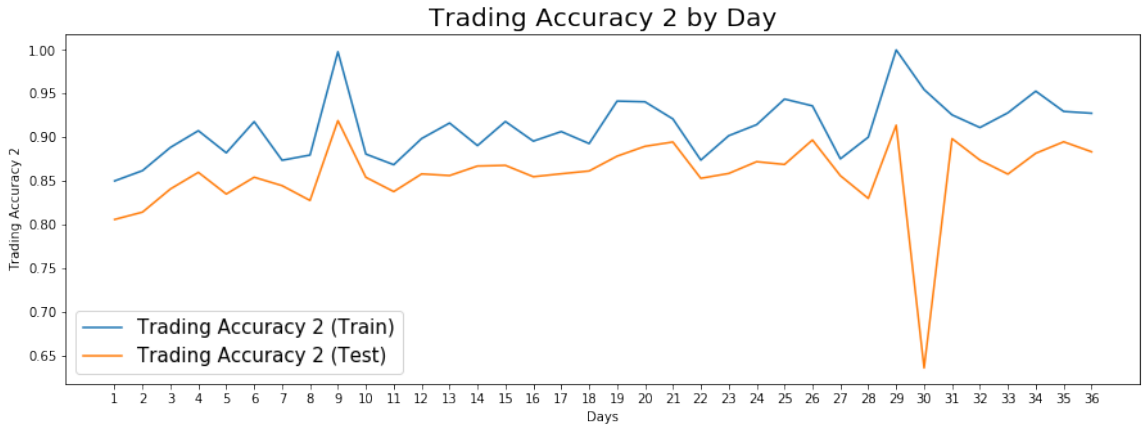

Figure 13: Random Forest Trading Accuracy 2 Over Time

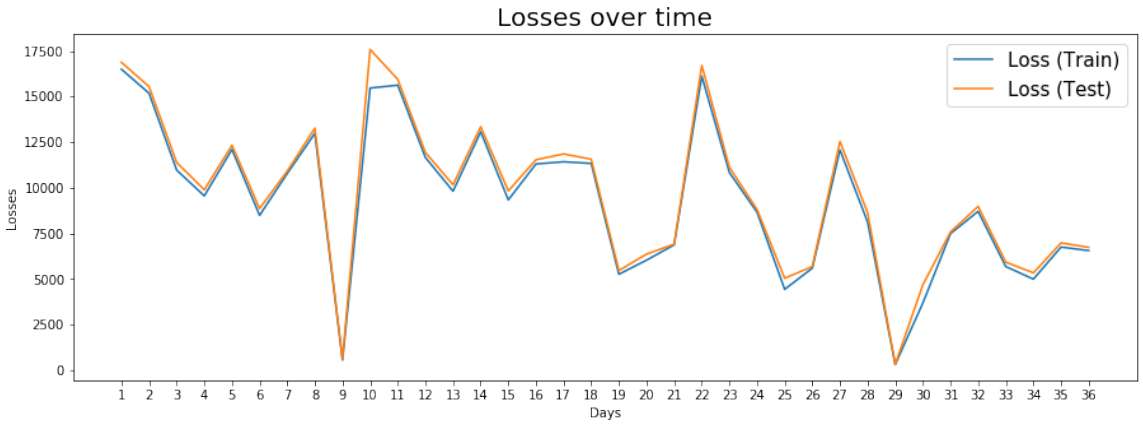

Figure 14: Random Forest XGboost Loss Over Time

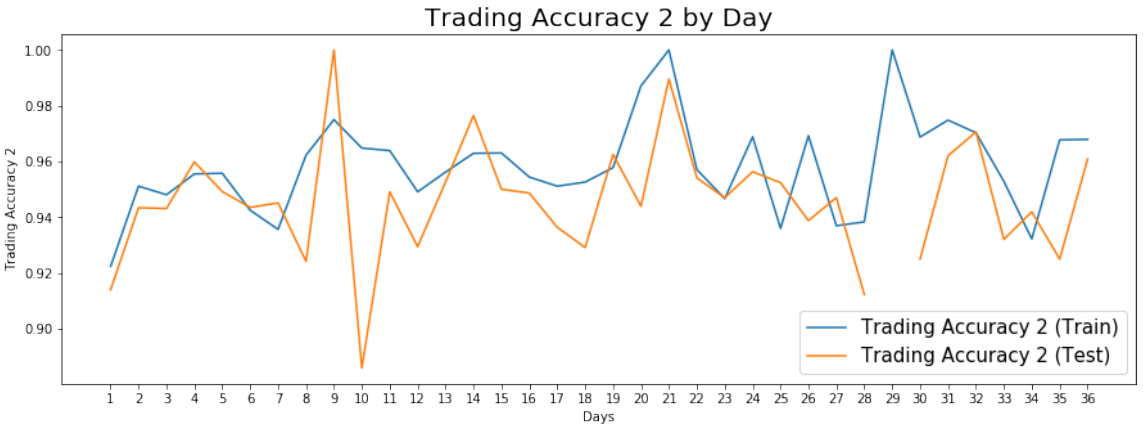

Figure 15: Random Forest XGboost Trading Accuracy 2 Over Time

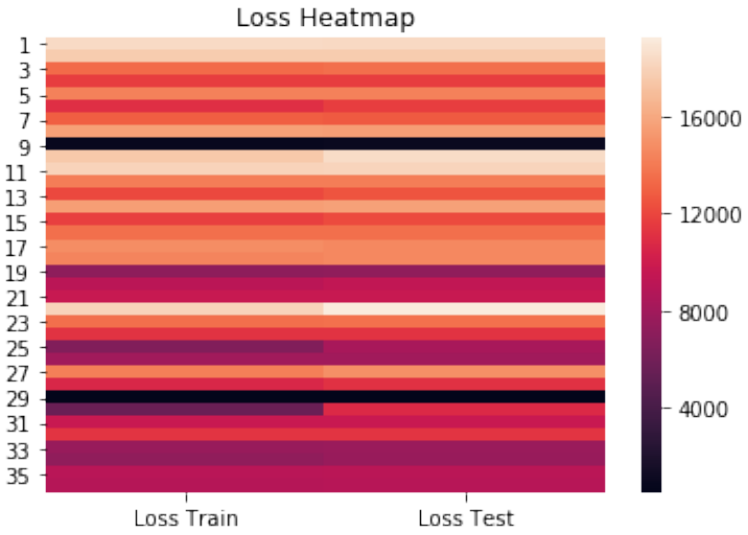

Figure 16: Linear Regression Loss Heatmap

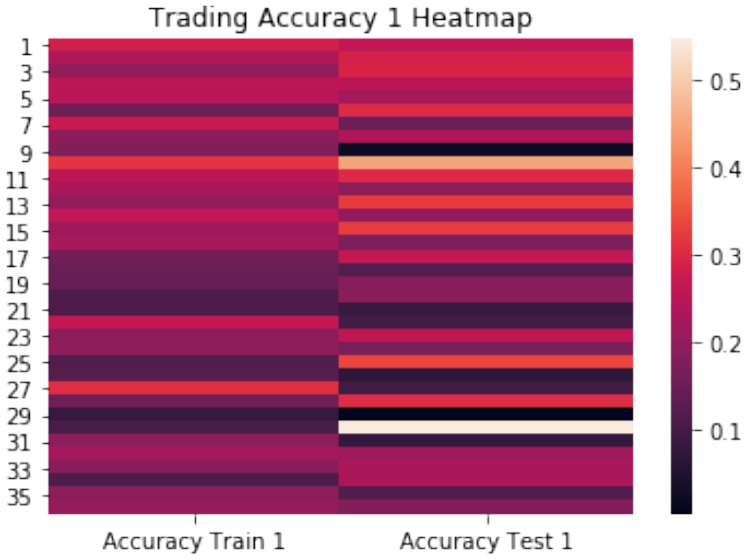

Figure 17: Linear Regression Trading Accuracy 1 Heatmap

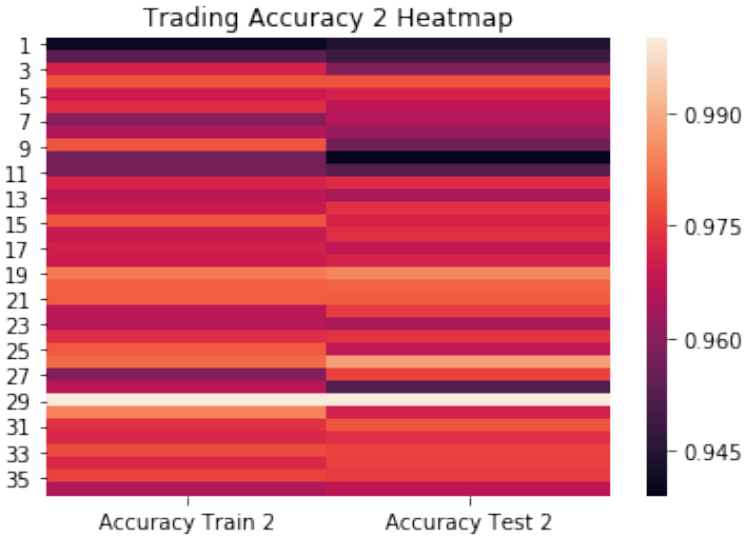

Figure 18: Linear Regression Trading Accuracy 2 Heatmap

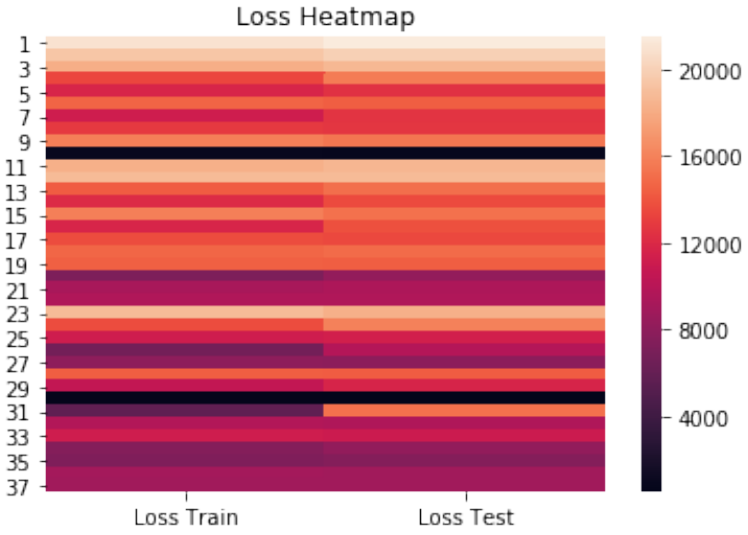

Figure 19: SVM Loss Heatmap

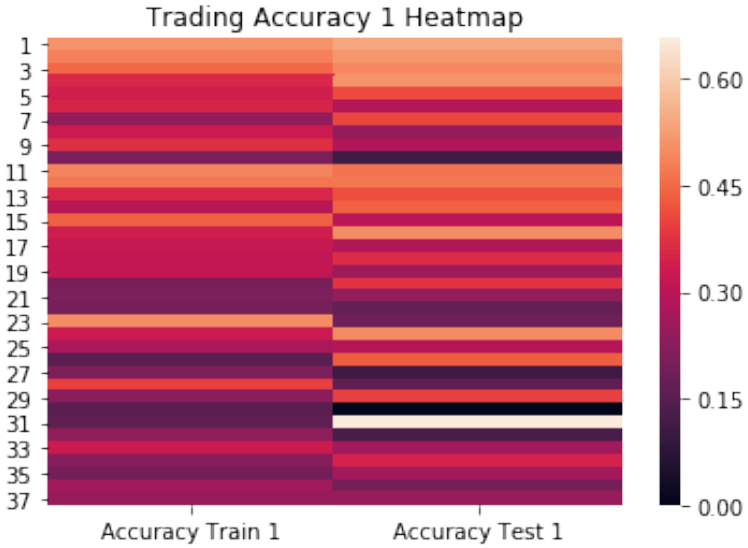

Figure 20: SVM Trading Accuracy 1 Heatmap

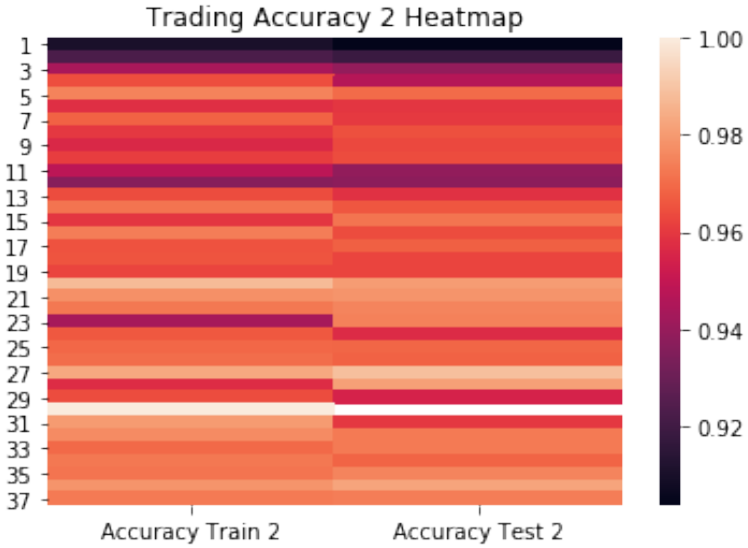

Figure 21: SVM Trading Accuracy 2 Heatmap

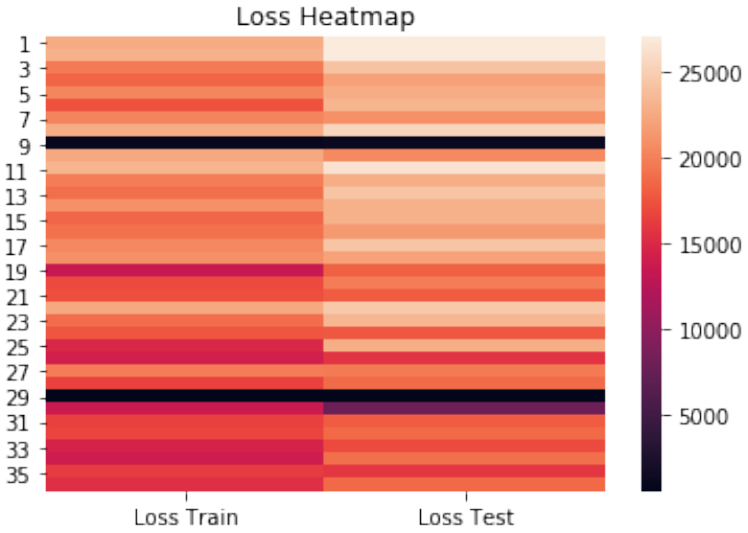

Figure 22: Random Forest Loss Heatmap

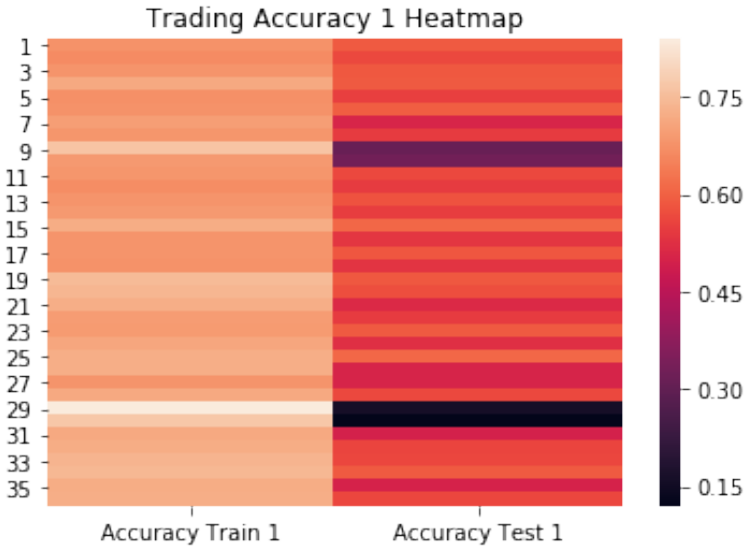

Figure 23: Random Forest Trading Accuracy 1 Heatmap

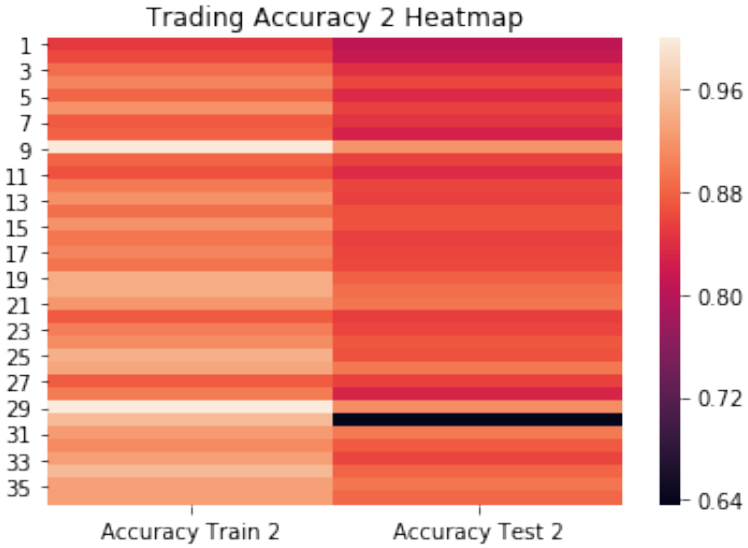

Figure 24: Random Forest Trading Accuracy 2 Heatmap

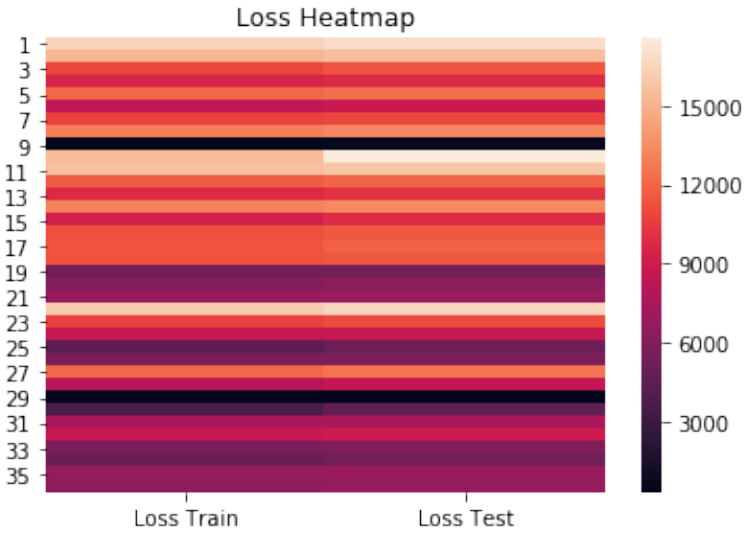

Figure 25: Random Forest XGboost Loss Heatmap

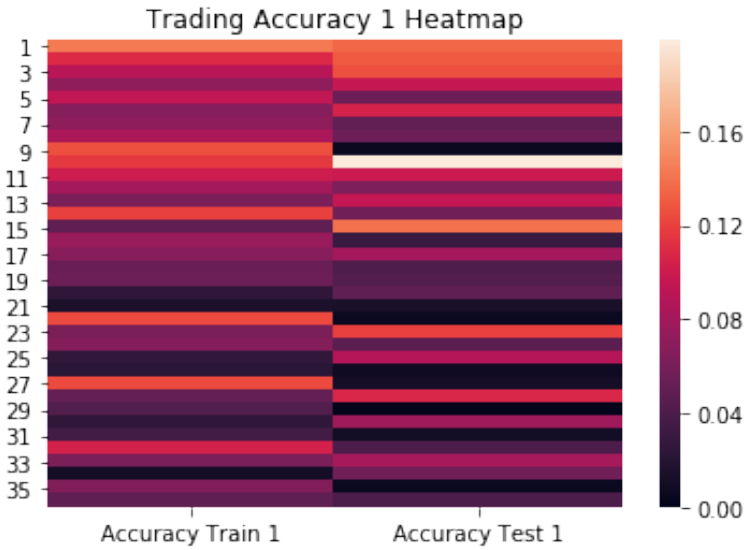

Figure 26: Random Forest XGboost Trading Accuracy 1 Heatmap

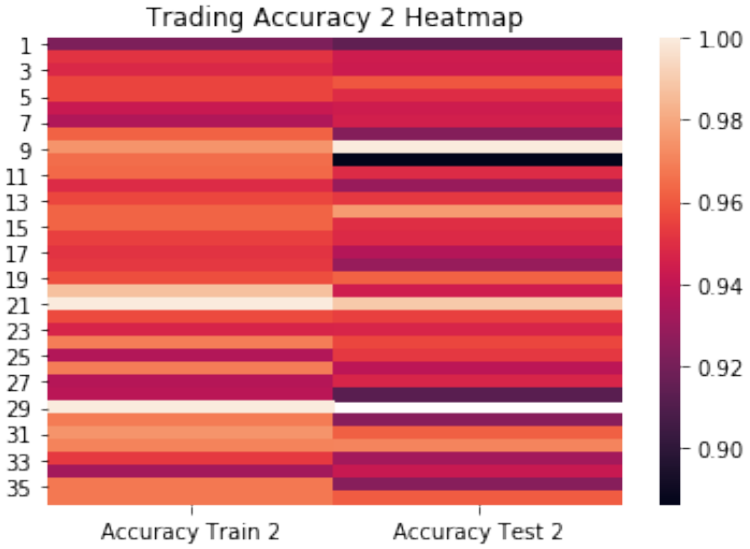

Figure 27: Random Forest XGboost Trading Accuracy 2 Heatmap

F Order Imbalance on Holidays

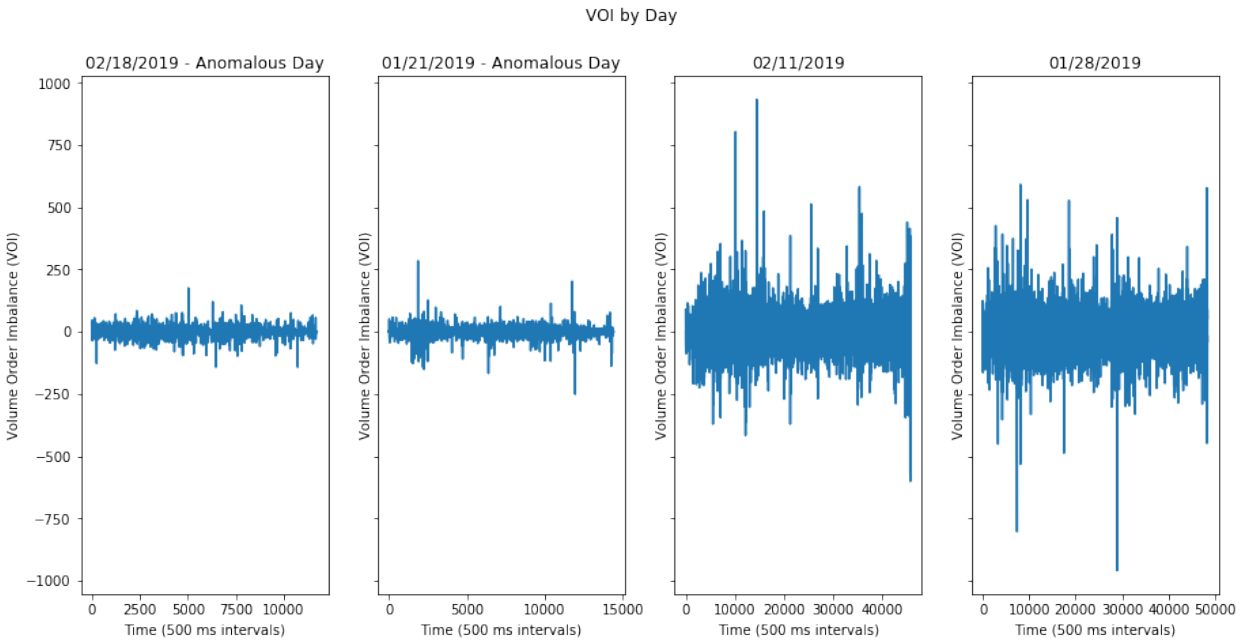

Figure 28: Holidays vs. Normal Trading Days

G Code

G.1 Data Cleaning

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tqdm import tqdm # wrap your iterable in tqdm() to see progress bar
from datetime import datetime

start = 8*3600*1000 # 8AM
end = 15*3600*1000 # 3PM


###################### Read in data ############################
fin = 'ESH2019'
ext = '.csv'
df = pd.read_csv(fin+ext) # input correct csv file
###################### ############ ############################
```

```
# drop unwanted columns
df.drop(['Price','Volume','Type'], axis = 1, inplace = True)

# NaNs occur in same row
df.dropna(inplace = True)

# midpoint
df['Mid'] = (df['Bid Price'].values + df['Ask Price'].values)/2

# mseconds
tmp = pd.to_timedelta(df['Time'].values)
tmp2 = (tmp.seconds*1000 + tmp.microseconds/1000)
df['msecs'] = np.float64((tmp2*2/1000)).round(0)*1000/2

# sec10
tmp = pd.to_timedelta(df['Time'].values)
tmp2 = (tmp.seconds)

# US market hours
df = df[(df['msecs'].values >= start) & (df['msecs'].values <= end)]

# take the latest bid-ask price for each 500 millisecond interval
df = (df.sort_values(by='Time', ascending=True)
  .groupby(['Date','msecs'])
  .head(1)
  .sort_index())

# generating each 500 ms interval for a day
msec_range = np.arange(start,end+.0001,500)
ndt = msec_range.shape[0]
ndate = np.unique(df['Date'].values).shape[0]

# each date within our data set
length = ndt*ndate
date = []
for idx, val in tqdm(enumerate(np.unique(df['Date'].values))):
   date += [val]*ndt

# creating the data frame with the correct dates and ms intervals
DF = pd.DataFrame({'Date':date,
              'msecs': list(msec_range)*ndate})

# merging the bid-ask data with new time intervals
DF = pd.merge(DF, df, how='left', on=['Date', 'msecs'])

# take out Sundays
dates = np.unique(DF['Date'].values)
day_p = np.zeros(DF.shape[0])
```

```
# weekends in dates
wknd = [day for day in dates if datetime.strptime(day,'%m/%d/%Y').weekday()>=5]
for idx, day in enumerate(wknd):
   day_p += DF['Date'].values == day
DF = DF[day_p == 0]

################################################################################

#M Measures
DF['dM']   = np.zeros(DF.shape[0])
DF['M_10ma'] = np.zeros(DF.shape[0])
DF['M_20ma'] = np.zeros(DF.shape[0])
DF['M_20fma'] = np.zeros(DF.shape[0])
DF['M_10fma'] = np.zeros(DF.shape[0])
DF['signal'] = np.zeros(DF.shape[0])
DF['signal2'] = np.zeros(DF.shape[0])
DF['signal3'] = np.zeros(DF.shape[0])
DF['signal4'] = np.zeros(DF.shape[0])
DF['signal5'] = np.zeros(DF.shape[0])
DF['signal6'] = np.zeros(DF.shape[0])


###
DF['B eq ind'] = np.zeros(DF.shape[0])
DF['B > ind'] = np.zeros(DF.shape[0])
DF['A eq ind'] = np.zeros(DF.shape[0])
DF['A < ind'] = np.zeros(DF.shape[0])
###
DF['dVB'] = np.zeros(DF.shape[0])
DF['dVA'] = np.zeros(DF.shape[0])
DF['OI'] = np.zeros(DF.shape[0])
###
DF['OI_a1'] = np.zeros(DF.shape[0])
DF['OI_a2'] = np.zeros(DF.shape[0])
DF['OI_a3'] = np.zeros(DF.shape[0])
DF['OI_a4'] = np.zeros(DF.shape[0])
DF['OI_a5'] = np.zeros(DF.shape[0])
###
DF['OI_5ma'] = np.zeros(DF.shape[0])
DF['OI_10ma'] = np.zeros(DF.shape[0])
DF['OI_20ma'] = np.zeros(DF.shape[0])

Q = .25 # minimum bid-ask spread

########### Generating the features #################

# iterate through each unique day in data set
for idx, val in tqdm(enumerate(np.unique(DF['Date']))):
```

```
i = DF['Date'] == val

# fill in NaN in price columns with most recent price
DF.loc[i,['Bid Price','Ask Price','Ask Size', 'Bid Size', 'Mid']] = (
      DF[i][['Bid Price','Ask Price','Ask Size', 'Bid Size','Mid']].interpolate(
      method='pad',axis=1,limit_direction='forward'))

################################################
# delta M Measures (Response Variable)
## change in mid price
DF.loc[i,'dM'] = DF.loc[i,'Mid']- DF.shift(1).loc[i,'Mid']

## 20 period moving average of mid price
DF.loc[i,'M_20ma'] =
    DF.loc[i,"Mid"].rolling(window=20,min_periods=20).mean().shift(1)

## 10 period moving average of mid price
DF.loc[i,'M_10ma'] =
    DF.loc[i,"Mid"].rolling(window=10,min_periods=10).mean().shift(1)

## 20 period forward moving average of mid price
DF.loc[i,'M_20fma'] =
    DF.loc[i,"Mid"].rolling(window=20,min_periods=20).mean().shift(-20)

## 10 period forward moving average of mid price
DF.loc[i,'M_10fma'] =
    DF.loc[i,"Mid"].rolling(window=10,min_periods=10).mean().shift(-10)


###### signals ######

## signal with 20 period moving averages (RESPONSE VARIABLE 1)
DF.loc[i,'signal'] =1*(DF.loc[i,'M_20fma']-DF.loc[i,'Mid']>=Q) +
    -1*(DF.loc[i,'M_20fma']-DF.loc[i,'Mid']<=-Q)

## signal with 10 period moving average (RESPONSE VARIABLE 2)
DF.loc[i,'signal2'] =1*(DF.loc[i,'M_10fma']-DF.loc[i,'Mid']>=Q) +
    -1*(DF.loc[i,'M_10fma']-DF.loc[i,'Mid']<=-Q)

## signal with mid/20p MA (RESPONSE VARIABLE 3)
DF.loc[i,'signal3'] =1*(DF.loc[i,'Mid']-DF.loc[i,'M_20ma']>=Q) +
    -1*(DF.loc[i,'Mid']-DF.loc[i,'M_20ma']<=-Q)

## signal with mid/10p MA (RESPONSE VARIABLE 4)
DF.loc[i,'signal4'] =1*(DF.loc[i,'Mid']-DF.loc[i,'M_10ma']>=Q) +
    -1*(DF.loc[i,'Mid']-DF.loc[i,'M_10ma']<=-Q)

## signal with 10f MA/ 10p MA (RESPONSE VARIABLE 5)
```

```
    DF.loc[i,'signal5'] =1*(DF.loc[i,'M_10fma']-DF.loc[i,'M_10ma']>=Q) +
        -1*(DF.loc[i,'M_10fma']-DF.loc[i,'M_10ma']<=-Q)

    ## signal with 20f MA/10p MA (RESPONSE VARIABLE 6)
    DF.loc[i,'signal6']=1*(DF.loc[i,'M_20fma']-DF.loc[i,'M_20ma']>=Q) +
        -1*(DF.loc[i,'M_20fma']-DF.loc[i,'M_20ma']<=-Q)
    ################################################

    ################################################
    # OI Measures
    DF.loc[i,'B eq ind'] = DF.loc[i,'Bid Price'] == DF.shift(1).loc[i,'Bid Price']
    DF.loc[i,'B > ind'] = DF.loc[i,'Bid Price'] > DF.shift(1).loc[i,'Bid Price']
    DF.loc[i,'A eq ind'] = DF.loc[i,'Ask Price'] == DF.shift(1).loc[i,'Ask Price']
    DF.loc[i,'A < ind'] = DF.loc[i,'Ask Price'] < DF.shift(1).loc[i,'Ask Price']

    ## change in ask/bid volumes
    DF.loc[i,'dVA']= (DF.loc[i,'A eq ind']*(DF.loc[i,'Ask
        Size']-DF.shift(1).loc[i,'Ask Size'])
                  +DF.loc[i,'A < ind']*DF.loc[i,'Ask Size'])
    DF.loc[i,'dVB']= (DF.loc[i,'B eq ind']*(DF.loc[i,'Bid
        Size']-DF.shift(1).loc[i,'Bid Size'])
                  +DF.loc[i,'B > ind']*DF.loc[i,'Bid Size'])

    ## order imbalance
    DF.loc[i,'OI'] = DF.loc[i,'dVB'] - DF.loc[i,'dVA']
    ################################################

    ################################################
    # OI Lags
    DF.loc[i,'OI_a1']=DF.loc[i,'OI'].shift(1)
    DF.loc[i,'OI_a2']=DF.loc[i,'OI'].shift(2)
    DF.loc[i,'OI_a3']=DF.loc[i,'OI'].shift(3)
    DF.loc[i,'OI_a4']=DF.loc[i,'OI'].shift(4)
    DF.loc[i,'OI_a5']=DF.loc[i,'OI'].shift(5)
    ################################################

    ################################################
    # OI Moving Averages
    DF.loc[i,'OI_5ma'] = DF.loc[i,"OI"].rolling(window=5,min_periods=5).mean()
    DF.loc[i,'OI_10ma'] = DF.loc[i,"OI"].rolling(window=10,min_periods=10).mean()
    DF.loc[i,'OI_20ma'] = DF.loc[i,"OI"].rolling(window=20,min_periods=20).mean()
    ################################################


# order imbalance ratio
DF['OI Ratio'] = ((DF['Bid Size'] - DF['Ask Size'])/
             (DF['Bid Size'] + DF['Ask Size']))
```

```
# fixed an issue where Order Imbalance values were objects
DF.loc[:,'OI'] = DF.loc[:,'OI'].astype(np.float64)
DF.loc[:,'Time'] = DF.loc[:,'Time'].replace(np.nan, '', regex=True)
```

G.2 Trading Results Function

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

def my_loss_summary(ytruetest, ytruetrain, ypredtest, ypredtrain):

   ytruetest=ytruetest[:-1]
   ypredtest=ypredtest[:-1]
   ytruetrain=ytruetrain[1:]
   ypredtrain=ypredtrain[1:]


   loss = np.array([[0,1,2],[1,0,1],[2,1,0]])
   days = len(ytruetest)
   losstrain = np.zeros(days)
   losstest = np.zeros(days)
   acctrain = np.zeros(days)
   acctest = np.zeros(days)
   acctrain1 = np.zeros(days)
   acctest1 = np.zeros(days)

   for i in range(days):
      conftrain = confusion_matrix(ytruetrain[i], ypredtrain[i])
      acctrain[i] = (conftrain[0,0] + conftrain[2,2])/np.sum(conftrain[0] +
          conftrain[2])
      acctrain1[i] = (conftrain[0,0] + conftrain[2,2])/np.sum(
         conftrain[0,0] + conftrain[0,2] + conftrain[2,0] + conftrain[2,2])
      losstrain[i] = np.sum(conftrain * loss)

      conftest = confusion_matrix(ytruetest[i], ypredtest[i])
      acctest[i] = ((conftest[0,0] + conftest[2,2])/
                np.sum(conftest[0] + conftest[2]))
      acctest1[i] = (conftest[0,0] + conftest[2,2])/np.sum(
         conftest[0,0] + conftest[0,2] + conftest[2,0] + conftest[2,2])
      losstest[i] = np.sum(conftest * loss)


   ind = [i+1 for i in range(days)]
   col = ['Loss Train','Loss Test','Accuracy Train 1',
        'Accuracy Train 2','Accuracy Test 1',
        'Accuracy Test 2']
```

```
plt.figure(figsize=(15,5))
plt.plot(ind, losstrain, label='Loss (Train)')
plt.plot(ind, losstest, label='Loss (Test)')
plt.title('Losses over time', fontsize=20)
plt.xlabel('Days')
plt.ylabel('Losses')
plt.xticks(np.linspace(1,days,days))
plt.legend(prop={'size':15})
plt.show()

plt.figure(figsize=(15,5))
plt.plot(ind, acctrain, label='Trading Accuracy 1 (Train)')
plt.plot(ind, acctest, label='Trading Accuracy 1 (Test)')
plt.title('Trading Accuracy 1 by Day', fontsize=20)
plt.xlabel('Days')
plt.ylabel('Trading Accuracy 1')
plt.xticks(np.linspace(1,days,days))
plt.legend(prop={'size':15})
plt.show()

plt.figure(figsize=(15,5))
plt.plot(ind, acctrain1, label='Trading Accuracy 2 (Train)')
plt.plot(ind, acctest1, label='Trading Accuracy 2 (Test)')
plt.title('Trading Accuracy 2 by Day', fontsize=20)
plt.xlabel('Days')
plt.ylabel('Trading Accuracy 2')
plt.xticks(np.linspace(1,days,days))
plt.legend(prop={'size':15})
plt.show()

retv = zip(col,(losstrain, losstest, acctrain, acctrain1, acctest, acctest1))
retdf = pd.DataFrame(dict(retv),index=ind)
retdf = retdf.round({'Loss Train': 0, 'Loss Test': 0, 'Accuracy Train 1': 4,
                'Accuracy Train 2': 4, 'Accuracy Test 1': 4, 'Accuracy Test 2': 4})

avedf = pd.DataFrame(retdf.mean())
avedf.columns = ['Averages']
avedf = avedf.round(4)

lossdf = retdf.iloc[:,0:2]
accudf = retdf.iloc[:,[2,4]]
accudf1 = retdf.iloc[:,[3,5]]

ax1 = sns.heatmap(lossdf)
ax1.set_title('Loss Heatmap')
plt.show()
ax2 = sns.heatmap(accudf)
```

```
    ax2.set_title('Trading Accuracy 1 Heatmap')
    plt.show()
    ax3 = sns.heatmap(accudf1)
    ax3.set_title('Trading Accuracy 2 Heatmap')
    plt.show()

    return (retdf, avedf)
```

G.3 Defining Scoring (Loss) Function

```
import numpy as np
from sklearn.metrics import confusion_matrix
from sklearn.metrics import make_scorer

def trading_loss(ytrue,ypred):
    conftrain = confusion_matrix(ytrue, ypred)
    loss = np.array([[0,1,2],[1,0,1],[2,1,0]])

    return np.sum(conftrain * loss)

trade_scorer=make_scorer(trading_loss,greater_is_better=False)
```

G.4 Cross Validating

G.4.1 Function to find indexes for cross validating

```
def my_train_test_idx(DF,xlabs=['OI_a1'],ylabs=['signal6'],return_df=False,n_days=4):
    df = DF[['Date']+xlabs+ylabs].dropna()

    #list of lengths of each day's data
    lengths = []
    for d in np.unique(DF['Date'].values)[:n_days]:
        lengths.append(df[df['Date']==d].shape[0])

    # where each day starts/ends
    bounds = np.cumsum(lengths)

    # list of 2-tuples of train & test indexes
    train_test_idx=[]
    prev=0
    for idx,val in enumerate(bounds[:-1]):
        train_id = np.arange(prev,val,1)
        test_id = np.arange(val,bounds[idx+1],1)
        prev = val
        train_test_idx.append((train_id,test_id))
```

```
    if return_df:
        return(train_test_idx,df)
    return (train_test_idx)
```

G.4.2 SVM

```
from sklearn.model_selection import GridSearchCV

# covariates to use
x_labs = ['OI_a1', 'OI_a2', 'OI_a3', 'OI_a4', 'OI_a5',
          'OI Ratio','OI_10ma','OI_20ma']

idx, df_cv = my_train_test_idx(DF,x_labs,return_df = True)

# cross validate the penalty parameter
tune_params = {'C':np.logspace(-5,5,20)}

cv_fit = GridSearchCV(svm.LinearSVC(dual=False,class_weight='balanced'),
                tune_params,scoring=trade_scorer,cv=idx)
cv_fit.fit(df_cv[x_labs].values,y=df_cv['signal6'].values)
```

G.4.3 Random Forest

```
x_labs = ['OI_a1', 'OI_a2', 'OI_a3', 'OI_a4', 'OI_a5',
          'OI Ratio', 'OI_5ma','OI_10ma','OI_20ma']
y_labs = ['signal6']

from sklearn.model_selection import RandomizedSearchCV, GridSearchCV

# Create the random grid
random_grid = {'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}

rf = RandomForestClassifier()
rf_cv = RandomizedSearchCV(rf, param_grid=param_grid,
                   scoring=trade_scorer,cv=idx)
rf_cv.fit(df_cv[x_labs],df_cv['signal6'])
```

G.4.4 Random Forest with Gradient Descent

```
import xgboost as xgb
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import TimeSeriesSplit

params={'max_depth': [3,6],
'min_child_weight': [1,4,8,16],
'eta': [0.05, 0.1, 0.3, 0.5, 0.7],
'gamma': [0, 3, 5],
'subsample': [0.1, 0.3, 0.5, 1],
'colsample_bylevel': [0.5, 0.7, 1]}


my_cv,training=my_train_test_idx(DF,xlabs=['OI_a1','OI_a2',
            'OI_a3','OI_a4','OI_a5','OI_5ma','OI_10ma',
            'OI_20ma','OI Ratio'],ylabs=['signal6'],
            return_df=True,n_days=4)

y_train=training.loc[:,'signal6']
X_train=training.loc[:,['OI_a1','OI_a2','OI_a3','OI_a4',
         'OI_a5','OI_5ma','OI_10ma','OI_20ma','OI Ratio']]

boost_fitRanCV = RandomizedSearchCV(XGBClassifier(silent=1),
            param_distributions=params, n_iter=20,
                scoring=trade_scorer,cv=my_cv).fit(X_train, y_train)
```

G.5 Linear Regression Stationarity Tests

G.5.1 ADF Test

```
import statsmodels
from statsmodels.tsa.stattools import adfuller

for i in range(test_dt.shape[0]):
   adf = adfuller(FDF.loc[FDF['Date'] == test_dt[i],:].dropna().loc[:,'OI_a5'])
   print (adf)

for i in range(test_dt.shape[0]):
   adf = adfuller(FDF.loc[FDF['Date'] == test_dt[i],:].dropna().loc[:,'OI_5ma'])
   print (adf)

for i in range(test_dt.shape[0]):
   adf = adfuller(FDF.loc[FDF['Date'] == test_dt[i],:].dropna().loc[:,'OI_10ma'])
   print (adf)
```

```
for i in range(test_dt.shape[0]):
   adf = adfuller(FDF.loc[FDF['Date'] == test_dt[i],:].dropna().loc[:,'signal7'])
   print (adf)
```

G.5.2 KPSS Test

```
from statsmodels.tsa.stattools import kpss

for i in range(test_dt.shape[0]):
   kp = statsmodels.tsa.stattools.kpss(FDF.loc[FDF['Date'] ==
       test_dt[i],:].dropna().loc[:,'OI_a5'])
   print (kp)

for i in range(test_dt.shape[0]):
   kp = statsmodels.tsa.stattools.kpss(FDF.loc[FDF['Date'] ==
       test_dt[i],:].dropna().loc[:,'OI_5ma'])
   print (kp)

for i in range(test_dt.shape[0]):
   kp = statsmodels.tsa.stattools.kpss(FDF.loc[FDF['Date'] ==
       test_dt[i],:].dropna().loc[:,'OI_10ma'])
   print (kp)

for i in range(test_dt.shape[0]):
   kp = statsmodels.tsa.stattools.kpss(FDF.loc[FDF['Date'] ==
       test_dt[i],:].dropna().loc[:,'signal7'])
   print (kp)
```

G.6 Testing

G.6.1 SVM

```
sv_fit = svm.LinearSVC(C=best_C,dual=False,class_weight='balanced')
for i,j in tqdm(test_idx):
   X_train = df_test.iloc[i,:][x_labs].values
   y_train = df_test.iloc[i,:]['signal6'].values
   X_test = df_test.iloc[j,:][x_labs].values
   y_test = df_test.iloc[j,:]['signal6'].values

   sv_fit.fit(X_train,y_train)

   # coefficients for SVM
   svm_coefs.append(sv_fit.coef_)
```

```
    # predictions
    p_tr = sv_fit.predict(X_train)
    p_te = sv_fit.predict(X_test)

    y_pred_vec_tr.append(p_tr)
    y_pred_vec_te.append(p_te)

    y_true_vec_tr.append(y_train)
    y_true_vec_te.append(y_test)

    # confusion matrices
    train_CFM.append(my_confusion_matrix(y_train,p_tr,[1,0,-1]))
    test_CFM.append(my_confusion_matrix(y_test,p_te,[1,0,-1]))

    # loss
    train_loss.append(trading_loss(y_train,p_tr))
    test_loss.append(trading_loss(y_test,p_te))
```

G.6.2 Random Forest

```
from sklearn.metrics import mean_absolute_error

DF = df
dates = DF['Date'].unique()
test_dt = dates[4:]

yhat_test_vector=[]
yhat_train_vector=[]
y_test_vector=[]
y_train_vector=[]
oob_error = []

for i in range(len(test_dt)-1):
    training=DF.loc[DF['Date'] == test_dt[i],:].dropna()

    y_train=training.loc[:,'signal6']
    X_train=training.loc[:,['OI_a1','OI_a2','OI_a3','OI_a4','OI_a5','OI_5ma','OI_10ma','OI_20ma']]

    test=DF.loc[DF['Date'] == test_dt[i+1],:].dropna()
    X_test=test.loc[:,['OI_a1','OI_a2','OI_a3','OI_a4','OI_a5','OI_5ma','OI_10ma','OI_20ma']]
    y_test=test.loc[:,'signal6']

    rf = RandomForestClassifier(n_estimators=100, min_samples_leaf=4,
                          min_samples_split = 5, max_depth = 10,
                          oob_score=True, class_weight='balanced')
```

```
    rf_fit = rf.fit(X_train.values,y_train)

    y_hat_test = rf_fit.predict(X_test.values)
    y_hat_train = rf_fit.predict(X_train.values)

    yhat_test_vector.append(y_hat_test)
    yhat_train_vector.append(y_hat_train)
    y_test_vector.append(y_test)
    y_train_vector.append(y_train)
    oob_error.append(1-rf_fit.oob_score_)
```

G.6.3 Random Forest with Boosting

```
clf = XGBClassifier(max_depth=3,
    min_child_weight=8,
    subsample=0.5,
    colsample_bytree=1,
    colsample_bylevel=0.7,
    n_estimators=100,
    gamma=3,
    eta=0.05)

yhat_test_vector=[]
yhat_train_vector=[]
y_test_vector=[]
y_train_vector=[]

for i in range(len(test_dt)-1):
    training=DF.loc[DF['Date'] == test_dt[i],:].dropna()

    y_train=training.loc[:,'signal6']
    X_train=training.loc[:,['OI_a1','OI_a2','OI_a3','OI_a4','OI_a5','OI_5ma','OI_10ma','OI_20ma','OI
        Ratio']]

    test=DF.loc[DF['Date'] == test_dt[i+1],:].dropna()
    X_test=test.loc[:,['OI_a1','OI_a2','OI_a3','OI_a4','OI_a5','OI_5ma','OI_10ma','OI_20ma','OI
        Ratio']]
    y_test=test.loc[:,'signal6']


    boost_fit = clf.fit(X_train.values,y_train)

    y_hat_test = boost_fit.predict(X_test.values)
    y_hat_train = boost_fit.predict(X_train.values)

    yhat_test_vector.append(y_hat_test)
```

```
    yhat_train_vector.append(y_hat_train)
    y_test_vector.append(y_test)
    y_train_vector.append(y_train)
```